



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

(Laboratorio di)  
**Amministrazione di sistemi**

# **Installazione e funzionamento di base di GNU/Linux**

**Marco Prandini**

Dipartimento di Informatica – Scienza e Ingegneria

# L'accesso all'hardware

- Il sistema operativo svolge una varietà di ruoli, sinteticamente
  - Astrazione delle risorse
    - Fisiche (*device*): storage, porte usb, schede di rete, ...
    - Logiche: filesystem, stack di rete, ...
  - Controllo dell'accesso alle risorse
    - Scheduling della CPU
    - Allocazione della memoria
    - Accesso a dispositivi
    - Gestione dei permessi
    - ...
- In Linux, queste funzioni sono realizzate attraverso un framework complesso per essere attivabili e disattivabili modularmente
- L'accesso all'hardware in particolare è astratto da moduli che implementano i *device driver*

# Device driver e kernel modules

- Alcuni driver sono cablati nel kernel, la maggior parte sono implementati da *moduli* del kernel dinamicamente caricabili
  - esplorare **/lib/modules**
  - approfondimento proposto: comandi **insmod**, **modprobe**, **lsmod**, **modinfo**
- Il codice del modulo definisce
  - come "farsi trovare"
  - come sono implementate le versioni specifiche di system call per il dispositivo gestito dal modulo
  - approfondimento proposto (hard): navigare nel codice di Linux <https://elixir.bootlin.com/linux/latest/source>



# Caricamento dei moduli

- Step 1: viene rilevato un dispositivo fisico
- Step 2: il controller I/O manda un interrupt
- Step 3: la CPU esegue l'interrupt handler, che è parte del kernel, che identifica l'evento e lo scrive su **dbus** (un canale pub-sub per tutti gli eventi di sistema)
- Step 4: **udev** riceve l'evento, e consultando **/lib/modules/<kernel\_version>/modules.alias** individua il modulo in grado di gestire il dispositivo

```
# modinfo psmouse
filename:
/lib/modules/4.13.0-37-generic/kernel/drivers/input/mouse/psmouse.ko
license:      GPL
description:  PS/2 mouse driver
author:       Vojtech Pavlik <vojtech@suse.cz>
srcversion:   16F6FEC23F72FA71FF21E33
alias:        serio:ty05pr*id*ex*
alias:        serio:ty01pr*id*ex*
depends:
intree:       Y
name:         psmouse
vermagic:     4.13.0-37-generic SMP mod_unload
signat:       PKCS#7
```

in ogni modulo sono dichiarate  
le stringhe identificative dei  
dispositivi fisici gestibili

**depmod** le raccoglie tutte e  
le scrive nel file **modules.alias**

# Funzionamento dei moduli

- Il modulo definisce in che modo vanno implementate le system call previste per la macro-categoria di dispositivi
  - **dispositivi a blocchi**: utilizzano buffer e cache per ottimizzare il trasferimento di blocchi di dimensione data,
    - si comportano un po' come un normale file, nel senso che "conservano" un elenco di byte singolarmente indirizzabili
    - es: dischi di vario tipo (ide, scsi, sata, usb, virtuali, ...)
  - **dispositivi a caratteri**: gestiscono il trasferimento dati un carattere/byte alla volta
    - possono consumare caratteri e farci qualcosa, ad esempio mostrarli su di un terminale
    - possono fornire caratteri se disponibili, o lasciare il consumatore in attesa se non ce ne sono, es. tastiera
    - in ogni caso non supportano la ricerca random (seek)
  - **dispositivi vari (misc)**



# Approfondimento - Un esempio di modulo

- Nelle due slide seguenti e nelle slide 10 e 11 sono riportati frammenti di un modulo molto semplice
  - non è un vero e proprio device driver
    - non tutti i moduli del kernel lo sono, in generale!
  - strutturalmente è identico
- Il modulo alloca nel kernel una variabile *counter*
  - implementa una propria versione di *write* che accetta qualsiasi stringa, la ignora, e incrementa counter
  - implementa una propria versione di *read* che restituisce il valore di counter
  - è quindi un semplice ma completo esempio di come le syscall consentono a codice user-space di accedere a dati e funzioni kernel-space
- Il codice è scaricabile da
  - <http://lia.disi.unibo.it/Courses/AmmSistemi1718/counter.tar>
  - include un file README con le indicazioni d'uso



# Implementazione delle system call

```
static ssize_t counter_read(struct file *file, char __user *buf, size_t
count, loff_t *ppos)
{
    unsigned char contents[COUNTER_BUFFER];
    unsigned i = *ppos;
    unsigned char *tmp;
    int size;

    size = scnprintf(contents, COUNTER_BUFFER, "%d", counter);

    for (tmp = contents; count-- > 0 && i < size; ++i, ++tmp)
        *tmp = contents[i];

    if (copy_to_user(buf, contents, tmp - contents))
        return -EFAULT;

    *ppos = i;

    return tmp - contents;
}
```

# Implementazione delle system call

```
static ssize_t counter_write(struct file *file, const char __user *buf,
size_t count, loff_t *ppos)
{
    unsigned char contents[COUNTER_BUFFER];

    if (count > COUNTER_BUFFER)
        return -EFAULT;

    if (copy_from_user(contents, buf, count))
        return -EFAULT;

    printk(KERN_INFO "Happily discarding message %s", contents);

    /* This is the only useful thing: to increase a line counter */
    counter++;

    return count;
}
```





# Funzionamento dei moduli

- Dov'è il codice di queste funzioni? In linea di principio c'è una tabella di puntatori, ad esempio

device	open	close	read	write	seek	...
psmouse	→	→	→	→	→	
counter	→	→	→	→	→	
disk	→	→	→	→	→	
...						

funzione che legge dal buffer della porta PS2 caratteri che rappresentano la posizione del mouse

funzione che incrementa counter per ogni riga ricevuta

funzione che comanda l'elettronica del disco per posizionarsi su di uno specifico blocco


- serve un sistema di nomi per dire "voglio invocare la *read* di XXX"!

# Device files

## ■ Nella cartella **/dev** si trovano molti file speciali

```
brw-rw---- 1 root    disk      8, 1 Mar 20 11:14 /dev/sda1
crw--w---- 1 las     tty       136, 0 Mar 20 11:17 /dev/pts/0
```



- ## ■ Sono punti di accesso alle periferiche, astratti come file
- sono memorizzati sul filesystem come normali inode, ma non hanno data block associati
  - su di essi, invocando le system call tipiche dei file si scatenano operazioni definite nel corrispondente *device driver*
  - quale d.d. usare è definito dal **major number**
  - l'istanza di dispositivo di quella classe è indicato dal **minor number**
  - possono essere di tipo **block** o **character**
- 

# Lo scheletro di un modulo

```
static const struct file_operations counter_fops = {  
    .owner      = THIS_MODULE,  
    .read       = counter_read,  
    .write      = counter_write,  
    .open       = counter_open,  
    .release    = counter_release,  
};
```

registrazione dei puntatori  
alle funzioni che  
implementano le syscall  
(fops=file operations)

```
static struct miscdevice counter_dev = {  
    COUNTER_MINOR,  
    "counter",  
    &counter_fops  
};
```

descrittore del driver:

- minor supportati
- **nome del file in /dev**
- **puntatore alle syscall**



# Lo scheletro di un modulo

```
static int __init counter_init(void)
{
    int ret;

    ret = misc_register(&counter_dev);
    if (ret) {
        printk(KERN_ERR "counter: can't misc_register on minor=
%d\n", COUNTER_MINOR);
    } else {
        counter = 0;
        printk(KERN_INFO "Useless string counter version 1.0\n");
    }
    return ret;
}

static void __exit counter_cleanup(void)
{
    misc_deregister(&counter_dev);
}

module_init(counter_init);
module_exit(counter_cleanup);
```

2. si esegue una  
funzione ...

3. che dice al kernel  
che descrittore usare  
(v. slide precedente)

1. al caricamento  
del modulo ...

# Device file → device driver

## ■ Quindi al caricamento del modulo

- il modulo dichiara che slot nella tabella dei puntatori vuole occupare, per mezzo di major e minor number
  - oppure chiede al kernel uno slot libero
- registra nelle celle i puntatori alle proprie implementazioni delle system call
- udev crea un device file col nome specificato dal modulo, a cui associa major e minor number

## ■ Ora qualsiasi operazione fatta sul device file scatena la specifica azione registrata dal device driver

1 **b**rw-rw---- 1 root disk <sup>2</sup>8,1 1 Mar 20 11:14 /dev/sda1

**open("/dev/sda1", O\_RDWR)**

- 1) il kernel riconosce sda1 come file speciale
- 2) ricava dall'inode major e minor
- 3) effettua il lookup nella tabella
- 4) esegue la funzione registrata

device	open	close	read	write	seek	...
10,1	→	→	→	→	→	
10,100	→	→	→	→	→	
3 <sup>2</sup> 8,1	4 →	→	→	→	→	
...						

# Device files di uso comune

- Alcuni device files non rappresentano vere periferiche, sono implementati dal kernel e sono utili per lavorare coi processi

**/dev/zero** produce uno stream infinito di zeri (binari)

**/dev/null** ogni read restituisce EOF, ogni write viene scartata

**/dev/random** produce byte casuali ad alta entropia  
→ bloccante se non ce n'è a sufficienza

**/dev/urandom** produce uno stream pseudocasuale illimitato



# Device files di uso comune

- Alcuni device files notevoli che rappresentano vere periferiche:

**/dev/tty\***

terminali fisici del sistema

**/dev/pts/\***

pseudo-terminali  
(dentro finestre del sistema grafico)

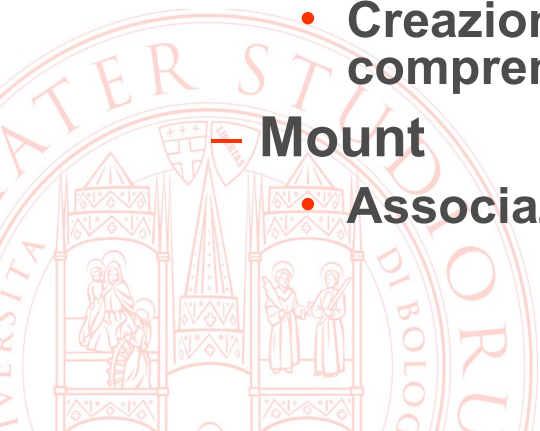
**/dev/sd\***

dischi e partizioni



# Il sistema di storage

- I dispositivi a blocchi rappresentano tipicamente supporti di storage
  - Reali: hard disk, SSD, usb drive, dischi ottici, ...
  - Logici: sistemi RAID, componenti LVM, ...
- Un dispositivo a blocchi è utilizzabile come un semplice elenco di blocchi dati di dimensione fissa, numerati
- Per renderlo fruibile servono tre operazioni
  - Partizionamento
    - Opzionale ma sempre utilizzato
  - Formattazione
    - Creazione dei metadati per organizzare lo spazio in modo comprensibile (filesystem)
  - Mount
    - Associazione dei singoli filesystem alla gerarchia di directory





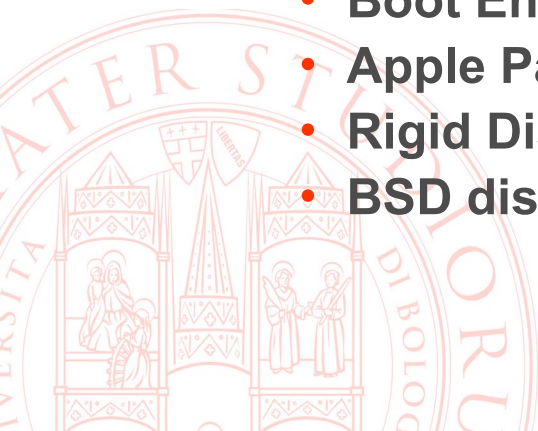
# Partizionamento

## ■ Suddivisione di un disco in sottoinsiemi di blocchi

- Ogni partizione si presenta come un dispositivo indipendente
- Utile per separare spazi con esigenze diverse
  - Di organizzazione (filesystem linux vs. Microsoft vs. ...)
  - Di persistenza (reinstallazione sistema vs. dati utente)
  - Di politiche di accesso (dati vs. programmi, ...)

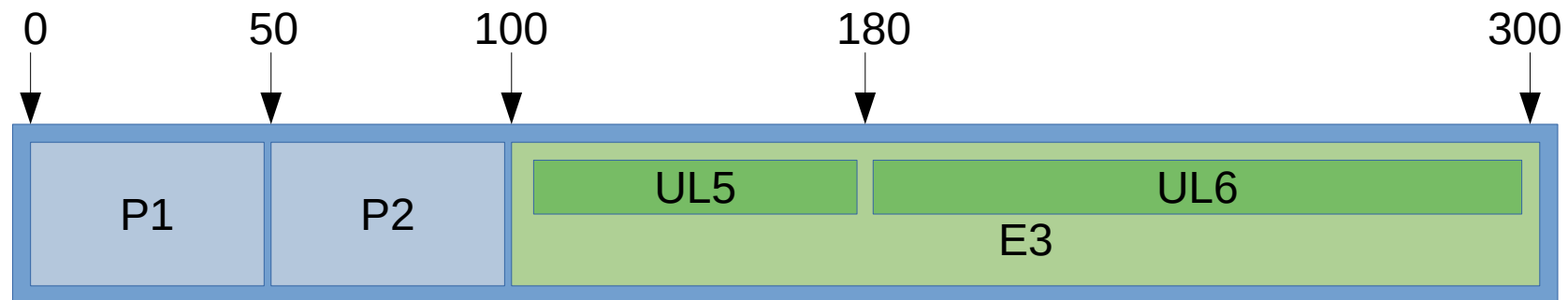
## ■ Vari standard

- Più diffusi su PC: Master Boot Record, GUID Partition Table
- Più rari o specifici di altre architetture
  - Extended Boot Record
  - Boot Engineering Extension Record
  - Apple Partition Map
  - Rigid Disk Block
  - BSD disklabel



# MBR-based

- **Tabella principale: max 4 entry (*partizioni primarie*)**
  - Semplicemente delimitate da blocco inizio – blocco fine (+ tipo)
- **Una di queste può essere contrassegnata *partizione estesa***
  - Lo spazio occupato dalla partizione estesa contiene un'ulteriore tabella
  - Nella tabella possono essere elencate fino a 12 *unità logiche*



*P1: primaria – inizio 0 – fine 50*

*P2: primaria – inizio 51 – fine 100*

*E3: estesa – inizio 101 – fine 300*

*UL5: unità logica – inizio 101 – fine 180*

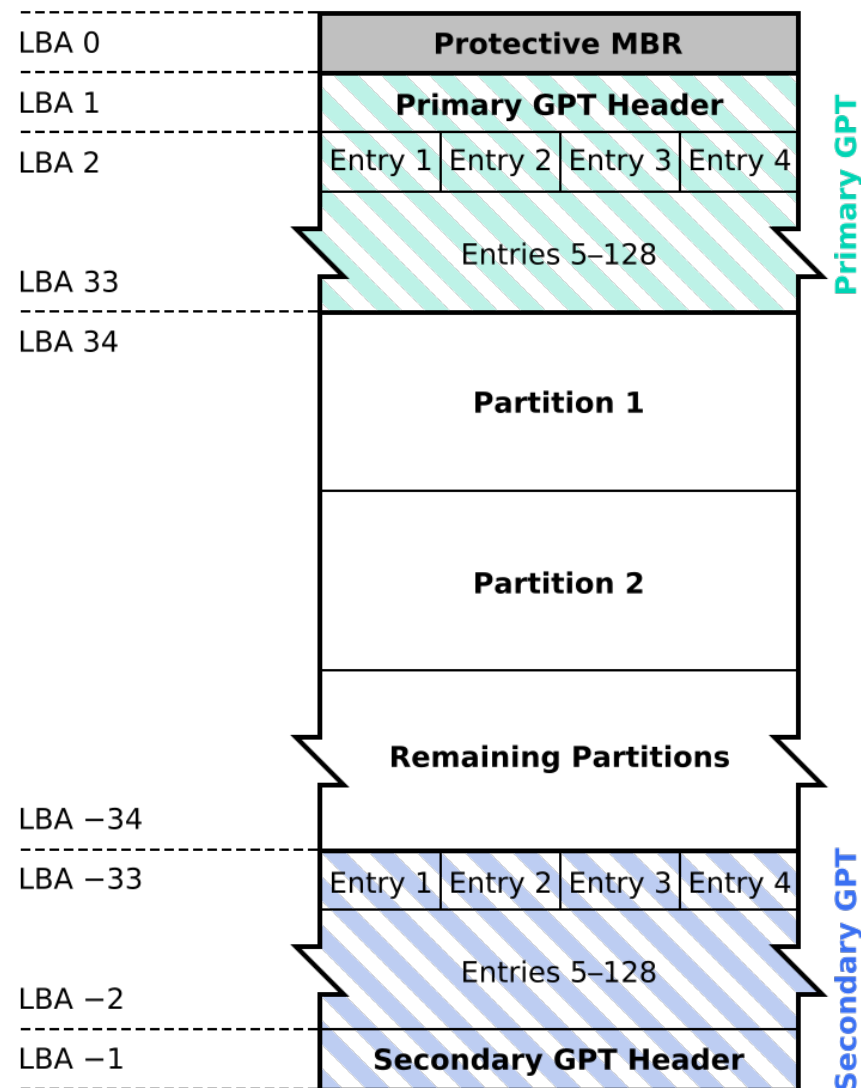
*UL6: unità logica – inizio 181 – fine 300*

- **Dal punto di vista dell'utilizzo, partizioni primarie e unità logiche sono equivalenti**

# GPT

- Utilizzato specialmente con UEFI, ma volendo anche con BIOS
- MBR max 15 partizioni di 2TiB
- GPT max 128 partizioni di 8 ZiB
- Oltre a limiti e tipo
  - Nome
  - GUID
  - Attributi

## GUID Partition Table Scheme



[https://en.wikipedia.org/wiki/GUID\\_Partition\\_Table#/media/File:GUID\\_Partition\\_Table\\_Scheme.svg](https://en.wikipedia.org/wiki/GUID_Partition_Table#/media/File:GUID_Partition_Table_Scheme.svg)

# Device file per dischi e partizioni

## ■ Comunemente unificati sotto il framework SCSI

/dev/sd~~XX~~~~NN~~ es. /dev/sda1

- ~~XX~~ = una o più lettere che identificano il “disco”
- ~~NN~~ = numero della partizione

## ■ Tipicamente i nuovi dischi NVMe/M.2 compaiono come

/dev/nvme~~X~~~~n~~~~Y~~~~p~~~~Z~~ es. /dev/nvme0n1p2

- ~~X~~ = identificatore del “disco”
- ~~Y~~ = identificatore del namespace
  - una sorta di macro-partizione hardware  
<https://nvmexpress.org/resources/specifications/>
- ~~Z~~ = numero della partizione

## ■ Gli identificatori di disco possono cambiare a seconda dell'ordine in cui i dischi vengono rilevati al boot!

# Criteri di partizionamento

## ■ Partizione di *swap* – memoria virtuale

- In origine consiglio 2xRAM, ma ora la penalità di prestazioni è inaccettabile per usarla davvero come memoria virtuale
- Uso comune sui laptop: partizione dedicata >>RAM per ibernazione
- Altri casi: allocare un po' di spazio solo per evitare che un piccolo esubero di uso di memoria mandi in crash il sistema
  - Possibilità sia di partizione separata che di file in partizione dati → file *sparse*

## ■ Collocazione dati – approccio minimale

- unica partizione con spazio sufficiente per tutto

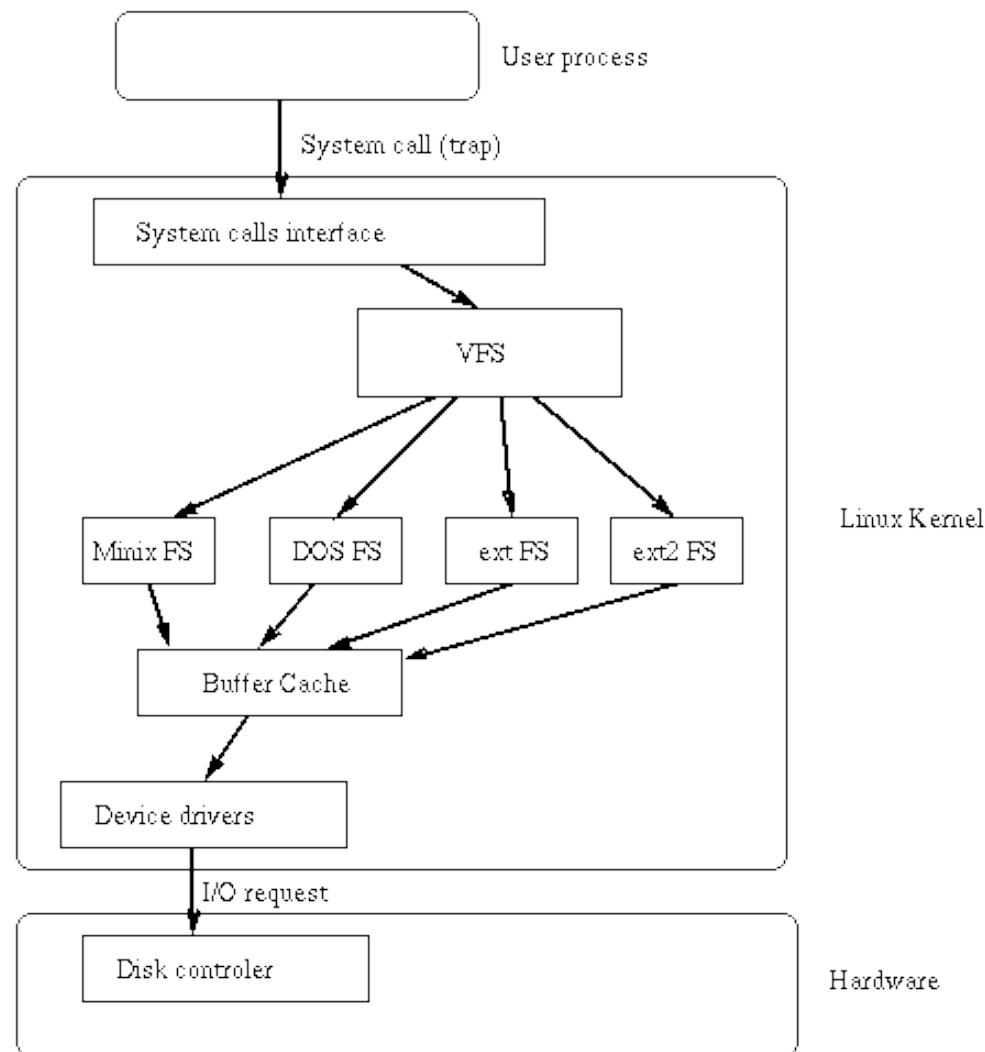
## ■ Collocazione dati per funzioni

- una partizione per ogni “tipo di accesso”, ad esempio:
  - una partizione per / (obbligatoria)
  - una partizione per i file di boot
  - una partizione per librerie e applicazioni → sola lettura
  - una partizione per code e log → alto traffico in lettura e scrittura
  - una partizione per aree utente → alto traffico e necessità di quota



# Formattazione

- **Crea il filesystem in una partizione**
  - Richiede spazio contiguo, può crescere o ridursi ma non può avere “buchi”
- **Permette l'accesso ai file secondo il modello del Virtual File System di Linux**
  - VFS astrae dall'organizzazione dei dati specifica di diversi FS



# ***Filesystem ext2***

- Il second extended filesystem è il più tradizionale in Linux.
- Non è journaled, e quindi anche se piuttosto robusto (e molto veloce) non è adatto a realizzare FS di grandi dimensioni, poichè il minimo guasto richiederebbe ore per la rilevazione e riparazione.
- Con blocchi di 4KB, le dimensioni massime sono
  - 2TiB per i file
  - 16TiB per l'intero filesystem
- La struttura è quella del FS Unix, con inodes che supportano fino a due livelli di indirettezza.
- Poichè le directory non sono indicizzate (la ricerca dei file avviene sequenzialmente) è bene non collocare più di 10-15000 file in una directory per non rallentare troppo le operazioni



# ***Filesystem ext3***

- **ext3 è nato per essere essenzialmente ext2 più un journal, mantenendo la compatibilità col predecessore. Per questo motivo non esibisce le prestazioni dei FS nativamente journaled, ma rispetto a ext2 ha due ulteriori vantaggi significativi:**
  - la possibilità di crescere, anche a caldo nelle ultime versioni
  - la possibilità di indicizzare le directory con htree, e quindi di gestire directory contenenti un maggior numero di file
- **Il journaling di ext3 può essere regolato su tre livelli:**
  - journal - registra sia i dati che i metadati nel journal prima del commit sul filesystem
  - ordered - registra solo i metadati, ma garantisce che ne sia fatto il commit solo dopo che i dati corrispondenti sono stati scritti sul FS
  - writeback - registra solo i metadati senza alcuna garanzia sui dati



# Filesystem ext4

- L'ultima evoluzione del filone “ext”, come sempre nata per offrire nuove feature pur consentendo un certo grado di retrocompatibilità, è stata introdotta in forma stabile nel kernel 2.6.28.
- Rispetto ad ext3, aumenta i limiti
  - di dimensione dei singoli file da 2 TiB a 16 TiB
  - di dimensione del filesystem da 16 TiB a 1 EiB
  - di numero di subdirectory da 32.000 a infinito
- Introduce i concetti di:
  - extent al posto del sistema di allocazione indiretta;
    - i file grandi sono allocati in modo contiguo anzichè essere suddivisi in moltissimi blocchi indirizzati in modo indiretto
  - allocazione dei blocchi a gruppi;
    - l'allocatore dei blocchi disco può essere usato per riservarne più di uno per volta
  - allocazione ritardata;
    - l'allocatore è invocato solo quando c'è l'effettiva necessità di scrivere su disco
  - journal checksumming
    - la consistenza del journal è ottenuta con un checksum invece che con una procedura di commit a due fasi, migliorando affidabilità e velocità

# Filesystem ext4 (continua)

- **Caratteristiche particolarmente interessanti per i sistemi embedded e real-time sono:**
  - **journal disattivabile**
    - elimina la causa delle maggiori lentezze di accesso ai dischi SSD e l'eccesso di scritture concentrate in pochi blocchi (usura);
  - **inode più grandi**
    - come sopra, poichè molti attributi estesi potranno essere ospitati direttamente nell'inode
  - **inode reservation**
    - aumenta il determinismo dei tempi di creazione dei file nelle directory
  - **persistent preallocation**
    - aumenta il determinismo dei tempi di scrittura dei dati in un file
- **Le caratteristiche di preallocation e reservation sono particolarmente importanti. Per contro, l'allocazione ritardata (di default) riduce la resistenza agli spegnimenti bruschi**
- **Dettaglio pratico: di default la formattazione ext4 riserva il 5% dei blocchi a root – è una strategia utilissima per evitare situazioni critiche, ma uno spreco per partizioni dati pure → disattivabile**

## ***Altri Journalled FS***

- **ReiserFS** - Il primo journaled FS ad essere inserito nel kernel di linux, è teoricamente molto performante su sistemi che gestiscono grandi quantità di file di piccole dimensioni, ma ha attratto critiche in merito alla sua stabilità. Inoltre lo sviluppatore capo ha imposto un modello molto conflittuale nei confronti della comunità, ed ha perso definitivamente credibilità per i suoi guai giudiziari.
- **XFS** - Sviluppato da Silicon Graphics per IRIX e **JFS** - Sviluppato da IBM per AIX, sono stabili, maturi, nativamente a 64bit (quindi su S.O. altrettanto a 64bit possono reggere partizioni da 8 ExaByte) ed offrono diverse ottimizzazioni molto vantaggiose; la loro scarsa diffusione è probabilmente dovuta solo alla consuetudine della maggior parte degli utenti Linux verso ext2/3

## ***Il futuro: btrfs?***

- **B-tree FS, abbreviato in btrfs e pronunciato “Butter F S”, è un progetto sviluppato da Oracle sotto licenza GPL.  
<http://btrfs.wiki.kernel.org/>**
- **Gli obiettivi sono quelli di integrare funzionalità per la scalabilità enterprise, simili a quelle offerte da ZFS di Sun:**
  - pooling di risorse HW, multi-device spanning
  - copy-on-write (versioning, writable snapshots, uso di supporti RO)
  - checksum su dati e metadati
  - compressione/deframmentazione/checking on line



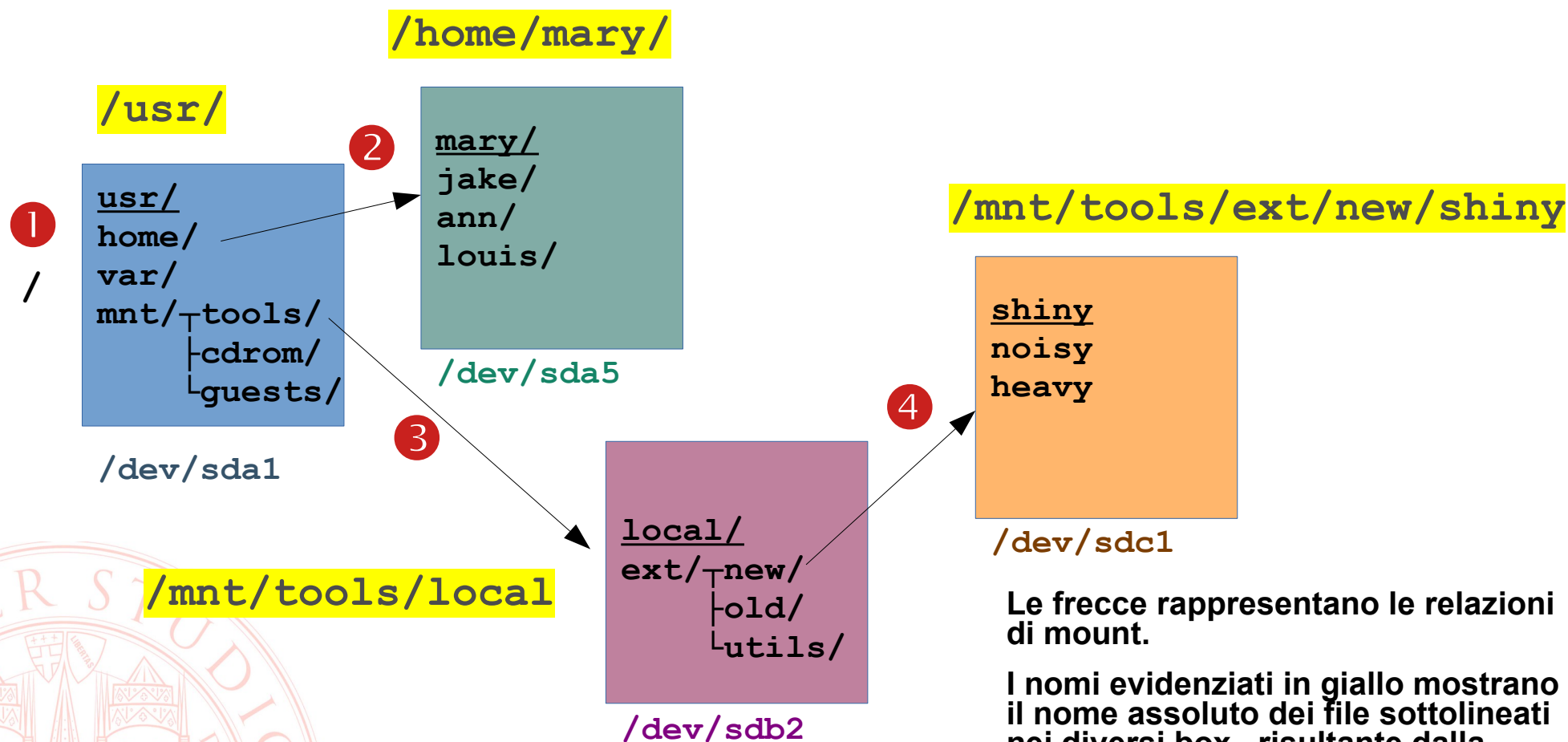
# Mount

- Il partizionamento crea una gerarchia locale di directory
- L'operazione di *mount* la “innesta” nella gerarchia globale



# Mount

- 1) `mount /dev/sda1 /`
- 2) `mount /dev/sda5 /home`
- 3) `mount /dev/sdb2 /mnt/tools`
- 4) `mount /dev/sdc1 /mnt/tools/ext/new`



# Il processo di avvio

- Per andare a regime il sistema attraversa un processo di boot, che può essere diviso in queste fasi:

- (1) BIOS – Individua i dispositivi di possibile caricamento del boot loader e l'ordine per esaminarli
  - Molti BIOS prevedono la possibilità di proteggere con password l'avvio o la modifica della configurazione
- (2) Boot Loader – Sceglie il sistema operativo e gli passa eventuali parametri
  - Gestione della “maintenance mode”
  - Stesso tipo di protezione con password come descritto per BIOS
- (3) Sistema operativo – carica i device driver (da non sottoestimare) e avvia il processo *init*
- (4) *init* – gestisce i *runlevel* o i *target* per coordinare l'inizializzazione del sistema, cioè avviare i servizi nell'ordine corretto





# UEFI e secure boot

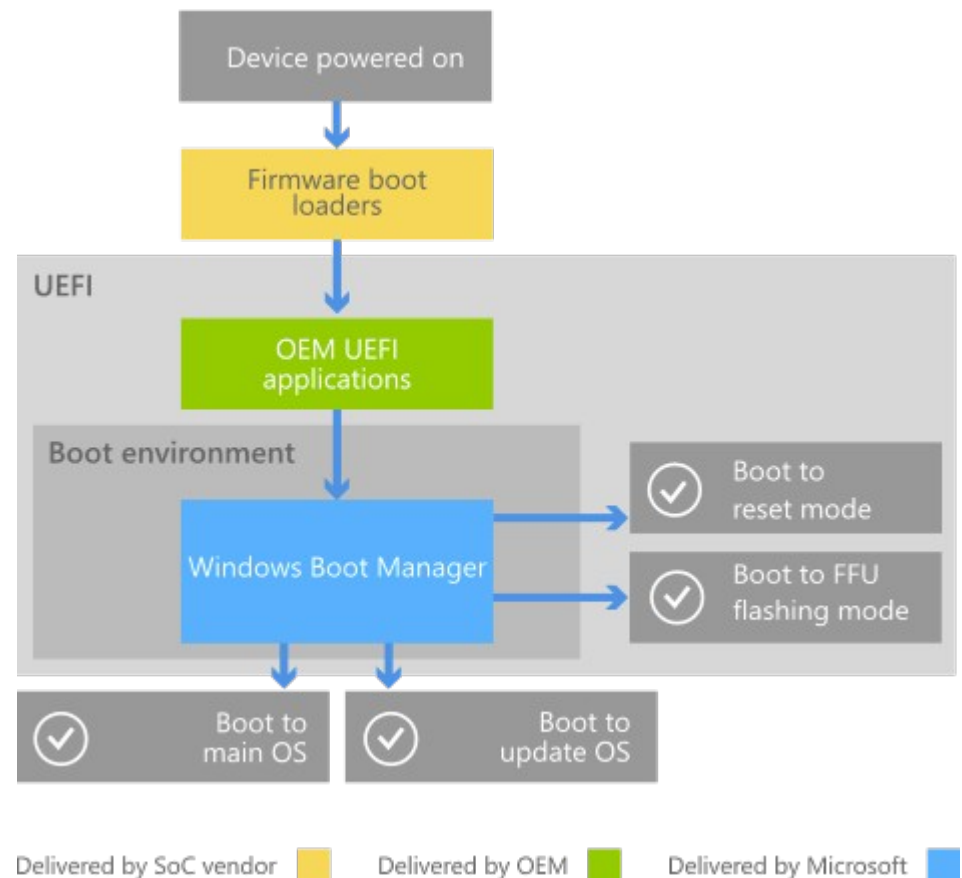
- EFI (Intel) nasce come interfaccia più flessibile del BIOS tra S.O. e firmware
- UEFI forum standardizza e aggiorna la specifica
- UEFI è un “mini OS”
  - milioni di righe di codice
  - standard per molte piattaforme
- UEFI verifica ogni componente software prima di passare il controllo al BootLoader e successivamente al sistema operativo
  - Richiede la disponibilità di un database di chiavi
  - Blocca il boot appena rileva una difformità
- I file di UEFI devono essere installati in una partizione dedicata contenente una gerarchia come in figura qui accanto

```
/EFI
  /Boot
  /Microsoft/
  /ubuntu
  /refind
```



# UEFI e i diversi sistemi

- Poiché UEFI richiede la verifica dell'autenticità del bootloader e del sistema operativo per mezzo di firme digitali depositate nell'hardware, inizialmente era consentito solo l'avvio di sistemi commerciali
- Linux Foundation ha ottenuto una chiave da Microsoft per autenticare un pre-bootloader in grado di autenticare a sua volta bootloader di sistemi aperti



[https://www.static.linuxfound.org/sites/main/files/lf\\_uefi\\_secure\\_boot\\_open\\_platforms.pdf](https://www.static.linuxfound.org/sites/main/files/lf_uefi_secure_boot_open_platforms.pdf)

# Dual boot

- È possibile istruire il boot loader a caricare sistemi diversi (o più versioni/configurazioni dello stesso sistema)
- Assumiamo che chi è interessato a installare Linux nativamente sul proprio PC per la prima volta abbia un'installazione di Microsoft Windows funzionante
- Vediamo in breve come procedere. Il risultato finale sarà un sistema che al boot permette di scegliere quale dei due sistemi operativi caricare
  - I sistemi saranno entrambi presenti sul disco ma non utilizzabili contemporaneamente
  - Nel momento in cui uno è in esecuzione, l'altro è come se non esistesse, le prestazioni sono quindi “native”



# Dual boot - installazione

## ■ I passi da svolgere saranno:

- Ridimensionare lo spazio dedicato a Windows per ricavare spazio non allocato
  - Verrà usato per creare una partizione (o più) dedicata a Linux
  - Nota: Windows non riconosce i tipi di partizione usati da Linux nè può accedere ai dati del filesystem ext4, ma non li “infastidisce”; Linux è in grado invece di accedere a filesystem di tipo FAT, FAT32, NTFS
- Scaricare un’immagine di un dispositivo avviabile con Linux
  - Ci sono innumerevoli alternative (approfondiremo in seguito)
  - Il file scaricato ha estensione .iso ed è una copia bit per bit di quanto si troverebbe memorizzato acquistando il “DVD originale”, ormai storico
- Creare un dispositivo ottico o USB con tale immagine
  - Nota: deve essere fatto con un software apposito, non copiando il file ISO!
  - Si devono sovrascrivere boot sector, tabella delle partizioni, filesystem
  - Tutto il contenuto del pendrive USB o del DVD viene cancellato
- Avviare il sistema da tale dispositivo
  - Il processo richiede di intervenire nella fase di caricamento del BIOS per scegliere il dispositivo esterno come fonte del bootloader
  - Può richiedere di intervenire sulle impostazioni UEFI per consentirne l’uso
- Seguire la procedura di installazione del sistema operativo

# Dual boot e alternative

- Non è l'unica soluzione, anche se è raccomandata a chi possa adottarla sul proprio computer
- Alternative:
  - Usare i PC del laboratorio
    - ☺ Linux è già disponibile
    - ☹ Non sono accessibili al di fuori delle ore del corso
  - Installare Linux come macchina virtuale
    - ☺ Non richiede tutta la procedura illustrata, bastano pochi click, e permette di continuare a usare il proprio sistema intanto che gira Linux
    - ☹ È più lento e potrebbe causare problemi in quanto all'interno della macchina che predisporremo gireranno a loro volta macchine virtuali
      - Per questo motivo non si può usare WSL (Windows Subsystem for Linux), il virtualizzatore integrato, che non funziona con alcuni pacchetti software che dovremo installare
- Questo è solo un riassunto dei principi. Un'intera sessione di laboratorio sarà dedicata all'installazione dell'ambiente per le esercitazioni nelle diverse modalità:
  - assistendo chi voglia predisporre il dual boot,
  - mostrando come utilizzare le macchine del laboratorio
  - illustrando come predisporre la macchina virtuale
- preparate tutto il necessario seguendo quanto specificato su Virtuale

# Collocazione delle risorse

- FHS (Filesystem Hierarchy Standard) definisce la struttura del filesystem Unix allo scopo di rendere più facile a programmi automatici ed utenti l'individuazione delle risorse, rendere più efficiente la condivisione di parti del filesystem e rendere più sicura la memorizzazione dei dati.

<http://www.pathname.com/fhs/pub/fhs-2.3.html>

- Le distinzioni base che guidano alla corretta collocazione dei dati in FHS sono 2:

	condivisibili	non condivisibili
statici	es. /usr /opt	es. /etc /boot
variabili	es. /var/mail /var/spool/news	es. /var/run /var/lock

# root directory (/)

- E' l'origine della gerarchia. Deve contenere tutti i dati necessari all'avvio del sistema, ma (storicamente) essere il più compatto possibile per ridurre i rischi di corruzione accidentale e poter essere alloggiato in media di scarsa capacità. Per rispondere a questi requisiti ed inoltre ospitare i punti iniziali di sottogerarchie più flessibili, deve contenere:

```
/ -- the root directory
+-bin      Essential command binaries
+-boot     Static files of the boot loader
+-dev      Device files
+-etc      Host-specific system configuration
+-lib      Essential shared libraries and kernel modules
+-mnt      Mount point for (temp) mounting a filesystem
+-opt      Add-on application software packages
+-sbin     Essential system binaries
+-tmp      Temporary files
+-usr      Secondary hierarchy
+-var      Variable data
```



# ROOT filesystem – alcuni componenti

`/dev`

La directory `/dev` contiene i file che costituiscono il punto di accesso, per i programmi utente, agli apparati connessi al sistema. Questi file sono essenziali per il funzionamento del sistema stesso.

`/etc`

La directory `/etc` è riservata ai file di configurazione locali del sistema. In `/etc` non devono essere messi eseguibili binari. I binari che in passato erano collocati in `/etc` devono andare in `/sbin` o `/bin`

`X11` e `skel` devono essere subdirectories di `/etc/`

```
/etc
|- x11
+- skel
```

La directory `X11` è per i file di configurazione del sistema X Window, come `XF86Config`. La directory `skel` è per i prototipi dei file di configurazione delle aree utente.



# ROOT filesystem – alcuni componenti

## `/lib`

La directory `/lib` deve contenere solo le librerie richieste per il funzionamento dei programmi che si trovano in `/bin` e `/sbin`.

## `/proc`

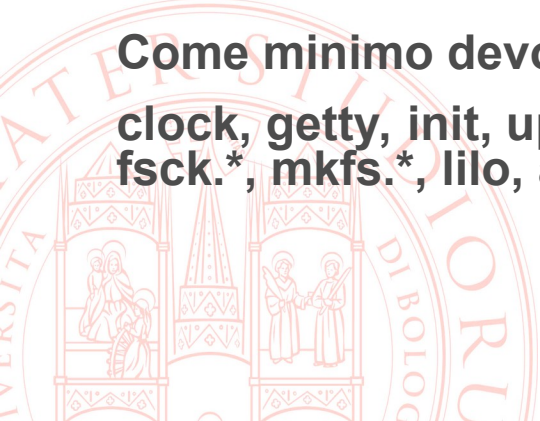
La directory `/proc` contiene file speciali che permettono di ottenere informazioni dal kernel o di inviare run-time informazioni al kernel, e merita di essere esplorata con attenzione.

## `/sbin`

La directory `/sbin` è riservata agli eseguibili utilizzati solo dall'amministratore di sistema, possibilmente solo quelli necessari al boot ed al mount del filesystem. Qualunque cosa eseguita dopo che `/usr` sia stato montato correttamente dovrebbe risiedere in `/usr/sbin` o in `/usr/local/sbin`

Come minimo devono essere presenti in `/sbin` i seguenti programmi:

`clock`, `getty`, `init`, `update`, `mkswap`, `swapon`, `swapoff`, `halt`, `reboot`, `shutdown`, `fdisk`, `fsck.*`, `mkfs.*`, `lilo`, `arp`, `ifconfig`, `route`





# /usr

La directory /usr è per i file condivisibili e statici. Risiede di preferenza su di una propria partizione, e dovrebbe essere montata read-only. Subdirs:

```
/usr
|- X11R6      X Window System
|- bin        eseguibili
|- dict
|- doc        documentazione diversa dalle man pages
|- etc        file di configurazione validi per il sito
|- games
|- include    C header files
|- info       GNU info files
|- lib        librerie
|- local
|- man        man pages
|- sbin       programmi linkati staticamente
|- share
+- src        codice sorgente
```



# VAR filesystem

La directory /var è riservata ai file non statici, sia condivisibili che non, ad esempio i file di log, di spool, di lock, di amministrazione e temporanei. Dovrebbe contenere le seguenti subdirs:

/var

- spool-----+-	at
- log	- cron
- catman	- lpd
- lib	- mail
- local	- mqueue
- named	- rwho
- nis	- smail
- preserve	- uucp
- run	+- news
- lock	
+- tmp	



# Filesystem virtuali

Esaminando un tipico sistema Linux si osservano diversi filesystem montati, che non hanno corrispondenza in alcun dispositivo fisico:

```
# mount
```

```
...  
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)  
proc on /proc type proc (rw,noexec,nosuid,nodev)  
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)  
udev on /dev type tmpfs (rw,mode=0755)  
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)  
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)  
fusectl on /sys/fs/fuse/connections type fusectl (rw)  
none on /proc/bus/usb type usbfs (rw,devgid=129,devmode=664)
```



# Filesystem virtuali principali

- Tra i filesystem virtuali notiamo:

**proc** e **sys**: permettono l'accesso diretto ai dati del kernel, quali

- aree di memoria
- parametri dei processi
- parametri di configurazione dei moduli

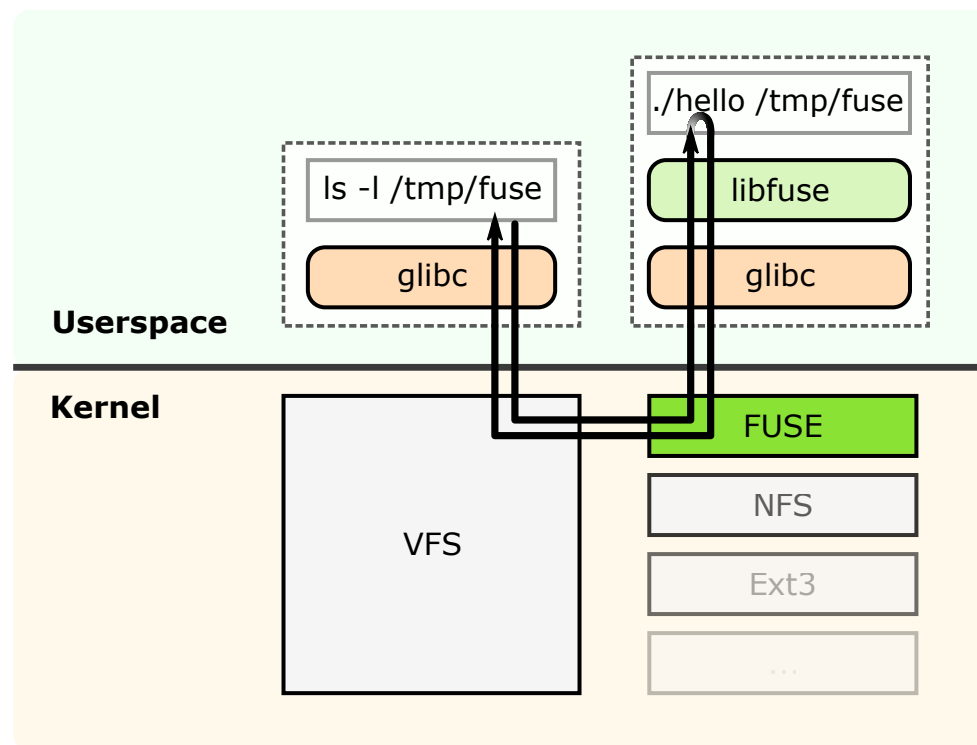
(vale la pena dare un'occhiata direttamente per rendersi conto di cosa è disponibile)

- **udev**: permette la generazione automatica da parte dei device drivers degli special file per l'accesso ai dispositivi
- **tmpfs** aree per la mappatura in memoria anzichè su disco di dati volatili



# FUSE

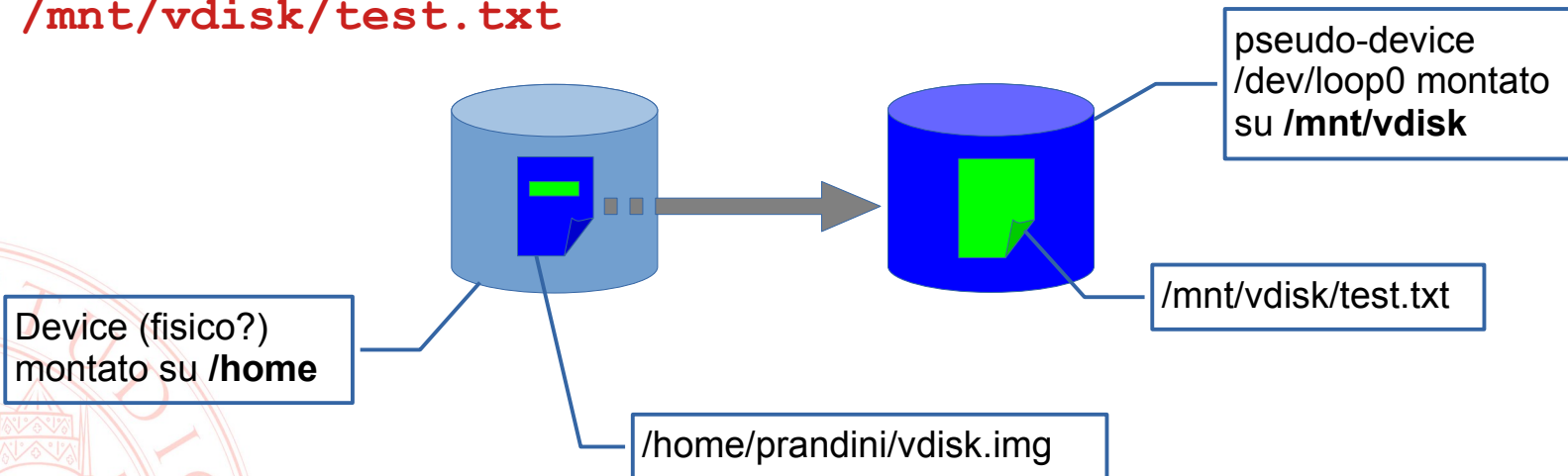
- Il driver FUSE (Filesystem in Userspace - <http://fuse.sourceforge.net/> ) offre un metodo per accedere a filesystem diversi da quelli previsti nel kernel, attraverso un processo che gira in user space, quindi senza modificare il kernel stesso.
- È tipicamente usato per accedere a filesystem che non necessitano di un vero e proprio accesso all'hardware, come quelli di rete.



# Loopback devices e mount

Tramite il comando `losetup` è possibile associare uno pseudo-device a blocchi ad un file: in questo modo è possibile utilizzare tutte le system call tipiche per l'accesso ai device a blocchi, che verranno mappate in operazioni di ricerca, lettura e scrittura dei byte del file anziché in comandi per un vero drive.

```
dd if=/dev/zero of=/home/prandini/vdisk.img bs=1024k count=100
losetup /dev/loop0 /home/prandini/vdisk.img
mkfs.ext3 /dev/loop0
mount /dev/loop0 /mnt/vdisk
touch /mnt/vdisk/test.txt
```



# Tuning

- Ogni filesystem dispone di numerosi parametri che influiscono sulle prestazioni e sulla robustezza; per la serie ext2/3/4 sono impostabili in uno o più dei modi seguenti:
  - alla creazione (`mkfs.ext[234]`)
  - al mount (`mount`)
  - a run time (`tune2fs`)
- Alcuni interventi comuni:
  - [dis]attivare il journal con `-O [^]has_journal`
  - [dis]attivare l'indicizzazione ad albero con `-O [^]dir_index`
  - ottimizzare le dimensioni di stride/stripe per device RAID
  - disattivare l'aggiornamento dell'access time con `-O noatime,nodiratime`
  - restringere l'utilizzo dei file (es. impedire la collocazione di device files o l'esecuzione di programmi sul filesystem)
  - abilitare estensioni per la sicurezza (cifratura, acl, bit speciali)





# Mount automatico delle partizioni

- L'associazione tra partizione e mount point è mantenuta nel file `/etc/fstab` perchè all'avvio del sistema il filesystem possa essere automaticamente predisposto

```
# <file system> <mount point> <type> <options> <dump> <pass>
/dev/sda1 / ext3 defaults,errors=remount-ro,relatime 0 1
/dev/sda4 /home ext3 defaults,relatime 0 2
```

